The Good and the Bad: Using Neuron Coverage as a DNN Validation Technique

Sujan Sai Gannamaneni, Maram Akila, Christian Heinzemann, and Matthias Woehrle

Abstract Verification and validation (V&V) is a crucial step for the certification and deployment of deep neural networks (DNNs). Neuron coverage, inspired by code coverage in software testing, has been proposed as one such V&V method. We provide a summary of different neuron coverage variants and their inspiration from traditional software engineering V&V methods. Our first experiment shows that novelty and granularity are important considerations when assessing a coverage metric. Building on these observations, we provide an illustrative example for studying the advantages of pairwise coverage over simple neuron coverage. Finally, we show that there is an upper bound of realizable neuron coverage when test data are sampled from inside the operational design domain (in-ODD) instead of the entire input space.

1 Introduction

In the past few years, there has been rapid growth in the usage of deep neural networks (DNNs) for safety-critical applications. Computer vision models, in particular, are being used in, e.g., autonomous driving or medical diagnostics. This shift in use cases is met by increasing demand for verification and validation (V&V) methods for DNNs [BEW+18, ZHML20, RJS+20]. However, unlike in traditional software

S. S. Gannamaneni (✉) · M. Akila
Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS,
Schloss Birlinghoven 1, 53757 Sankt Augustin, Germany
e-mail: sujan.sai.gannamaneni@iais.fraunhofer.de

M. Akila
e-mail: maram.akila@iais.fraunhofer.de

C. Heinzemann · M. Woehrle
Robert Bosch GmbH, Corporate Research, Robert-Bosch-Campus 1,
71272 Renningen, Germany
e-mail: christian.heinzemann@de.bosch.com

M. Woehrle
e-mail: matthias.woehrle@de.bosch.com

T. Fingscheidt et al. (eds.), *Deep Neural Networks and Data for Automated Driving*,
https://doi.org/10.1007/978-3-031-01233-4_14

engineering, developing rigorous techniques for V&V of DNNs is challenging as their intrinsic working is not directly of human design. Instead, the relevant parameters are learned, and their meaning often eludes the human engineer.

With the strengthened focus on V&V for DNNs, there was a gradual shift away from mere performance metrics to developing techniques that measure and investigate, e.g., (local) interpretability or robustness. While insightful, such approaches focus on specific instances of a given (test) dataset. However, they typically do not detail the required number of elements in those datasets in the sense of sufficient testing.

Inspired by testing in software engineering, Pei et al. proposed a variant of code coverage called neuron coverage (NC) [PCYJ19]. In analogy to checking if a line of code is executed for any given test sample, NC provides us with the percentage of activated neurons for a given test dataset.

For a test (or in this case: coverage) metric, the granularity or "level of difficulty" is crucial; if a test is too easy to fulfill, the chances are low that it will uncover errors. If, on the other hand, it is barely possible (or even impossible) to fully perform the test, it becomes ill-defined as no stopping criterion exists or can be found (cf. [SHK+19]). This makes the level of granularity of a test a decisive criterion. We can illustrate this on the example of classical code coverage for the pseudo-code shown in Algorithm 1.

Algorithm 1 Code coverage example

```
1: Set a
2: c ← 2
3: if a > 2 then
4:     c ← −4
5: end if
6: if a < 4 then
7:     a ← c · a
8: end if
9: return a + c
```

It takes a (real) input variable a and maps it onto a (real) output. Following code coverage requirements, we would need to find inputs a_i such that in total, each line of code is executed at least once. The two values $\{a_1 = 1, a_2 = 5\}$ would lead the algorithm to either pass through the first or the second if-clause, respectively, and would thereby together fulfill this requirement. The advantage of this test is that, besides being simple, it has a well-defined end when the full code is covered. Conversely, should not all of the code be coverable, it directly revealed unreachable statements that can be removed from the code without changing its functionality. The latter would be seen as a step towards improving the code's "readability" and as avoiding potentially unintended behavior, should those parts of code be reachable by some unforeseen form of a statement. However, this form of coverage can provide only a rudimentary test of the algorithm. For instance, the output in the example differs significantly if the algorithm runs through both clauses simultaneously, which would be the case, e.g., for $a_3 = 3$. We cannot (or at least not with a reasonable amount

of effort) test a given code for all potential eventualities. However, we can evaluate with more complex test criteria, pair-wise tests in the place of the single-line tests. A reasonable approach could be to test if both if-clauses "A" and "B" were used in the same run or whether only one of them was passed. While this approach would surely have uncovered the potentially erroneous behavior from the third input a_3, it also scales quadratically with the number of clauses, while the previous criterion scaled linearly. Additionally, it is a priori unclear to which extent an algorithm can be covered in principle. For instance, in the example, no value for a can be found such that both if-clauses would be skipped simultaneously.

The above example on algorithmic code coverage can be transferred to the coverage of neurons within a DNN. Assuming, as we will also do in the following, an architecture based on ReLU non-linearities, where $\text{ReLU}(y) = \max(y, 0)$, we may count a unit (neuron) as covered, if, for a given input, its output after application of the ReLU is non-zero.[1] Full coverage, for this measure, would thus be achieved if a given test set $\mathcal{X}$ can be found such that for any unit of the DNN, at least one data sample $\mathbf{x}_i \in \mathcal{X}$ exists where that unit is covered. As with the code coverage before, the inability to construct or find such a sample $\mathbf{x}_i$ could indicate that the unit in question is superfluous and might be removed (pruned) without changing the functionality of the DNN. The generalization to pair-wise metrics is straightforward but owing to the often sequential structure of feed-forward DNNs, a further restriction to adjacent layers is considered, i.e., pairs of only adjacent layers are considered when evaluating with this metric. There are, however, two important distinctions to code coverage. At first, interaction among layers, although reminiscent of the single control flow in the above code example, is significantly more complicated. Examining Algorithm 1 more closely, the interaction was mediated by a latent variable c. To some extent, each unit within a layer can be seen as such a variable allowing for various behaviors in the subsequent layer. Moreover, not activating a unit is not an omission but a significant statement in its own right, compare for instance [EP20]. As a second point, input parameters and variations in classical software are often reasonably understood. For DNNs, on the contrary, often their use is motivated by the inability to specify such variations. Therefore, a secondary use of coverage metrics to determine *the novelty of a data point* $\mathbf{x}_j$ w.r.t. a test set $\mathcal{X}'$ ($\mathbf{x}_j \notin \mathcal{X}'$) becomes apparent by measuring how strongly the coverage values between $\mathcal{X}'$ and $\mathcal{X}' \cup \{\mathbf{x}_j\}$ differ. Such a concept is based on the assumption that coverage, to some degree, represents the internal state (or "control flow" if seen from a software point of view) of the DNN and could thus be used to determine the completeness of the test set in the sense that no further novel states of the DNN could be reached.

In this chapter, we address the following aspects: On the level of single unit coverage, we investigate the effect of granularity, i.e., to which degree layer-wise resolved metrics allow for a more detailed test coverage. In this context, we also investigate whether the coverage metric is informative regarding the novelty of samples by comparing trained and untrained DNNs on CIFAR-10. For both single and pairwise

[1] While generalizations to non-zero thresholds exist, see for instance [MJXZ+18], we restrict ourselves to this case as it is a natural choice for ReLU non-linearities.

coverage, we address under which circumstances (full) coverage can be reached. In a (synthetic) toy example, we also differentiate how strongly these statements depend on the restrictions posed on the test set data when staying within a DNN's intended input domain (i.e., the operational design domain, ODD).

2 Related Works

Pei et al. [PCYJ19] introduced neuron coverage, i.e., looking at the percentage of neurons activated in a network, as a testing method. Using a dual optimization goal, they generate realistic-looking test data that increases the percentage of activated neurons. The method proposed in [MJXZ+18], a variant of [PCYJ19], uses the same definition of NC and additionally introduces k-multisection coverage and neuron boundary coverage. Another variant, [TPJR18], uses image transformations on synthetic images to maximize neuron coverage.

Sun et al. [SHK+19], inspired from modified condition/decision coverage (MCDC) in the software field, proposed four condition decision-based coverage methods. Furthermore, they showed that the proposed methods subsumed earlier works, i.e., satisfying their proposed methods also satisfies the weaker earlier coverage methods.

However, there are also several works expressing criticism of NC as a validation technique [DZW+19, HCWG+20, AAG+20]. While in [DZW+19], it is shown that there is limited to no correlation between robustness and coverage, Harel-Canada et al. [HCWG+20] show that test sets generated by increasing NC, as defined in [PCYJ19, TPJR18], fail in specific criteria such as defect detection, naturalness, and output impartiality. Abrecht et al. [AAG+20] showed that there was a discrepancy in the way how neurons in different layers were evaluated in earlier works due to differing definitions of what constitutes a neuron in a DNN. Furthermore, they show that evaluating NC at a more granular level provides more insight than just looking at coverage over the entire network. However, they also show that even this granular coverage is still easy to achieve with simple augmentation techniques performing better than test data generation methods like DeepXplore [PCYJ19].

Abrecht et al. [AGG+21] presented an overview of testing methods of computer vision DNNs in the context of automated driving. This includes a discussion of different types of adequacy criteria for testing and concretely of structural coverage for neural networks as one form of an adequacy criterion that supports testing.

In this chapter, we follow the definitions of layer-wise coverage (LWC) [AAG+20] to determine NC and adjust it for pairwise coverage (PWC). While the fundamental definition, i.e., a neuron or a unit is considered as covered as long as there is at least one sample $\mathbf{x}_i$ in the test set $\mathcal{X}$ such that the unit produces a non-zero output, remains unchanged across definitions, the precise meaning of what constitutes a unit differs. Compared to other metrics, e.g., DeepXplore [PCYJ19], LWC provides a more fine granular resolution of coverage, especially concerning convolutional layers. While one could treat each channel of the convolutional layer as a single unit with respect

to coverage, LWC demands that each instance of the use of the filter is counted as an individual unit to be covered. In terms of an example, if a two-channel convolutional layer results in an output of $3 \times 3 \times 2$ real numbers, where the first two numbers are due to the size of the input image, LWC would require to cover each of the 18 resulting "neurons" while DeepXplore would consider only two "neurons", one for each channel.

3 Granularity and Novelty

Several works [SHK+19, AAG+20] have shown that achieving high NC is trivial with only a few randomly selected in-domain non-adversarial images. Abrecht et al. [AAG+20] have further shown that trained and untrained networks show similar NC behavior for a given test dataset. Intuitively, one would expect that, first, a metric used to investigate the safety of a DNN in terms of finding new input samples should not be easily satisfiable and, second, that the metric should depend on the "quality" of the input. Typically, a DNN is trained to solve a specific problem that only covers a very small subset of all possible inputs, e.g., classifying an image as either showing a cat or dog could be contrasted by completely unfitting inputs such as the figures in this chapter. Following the analogy of code coverage, this would be the difference between expected and fully random (and mostly unfitting) inputs, where for the latter, one would expect nonsensical outputs or error messages. Especially for neural networks, this second point is of relevance as they are typically used in cases where a distinction between reasonable and unreasonable input can be made only by human experts or DNNs but not via "conventional" algorithms. This makes an extension of the input coverage to novel examples challenging, as "errors" discovered on erroneous inputs might lead to questionable insights into the DNNs trustworthiness or reliability.

Within this section, we elaborate on the two raised concerns by investigating an example DNN on the level of (single-point) NC. For this, we will look not only at the reached coverage but take a closer look at its saturation behavior. Further, we compare both trained and untrained (e.g., randomly initialized) versions of the DNN using either actual test data (from the dataset) or randomly generated Gaussian input. To be more precise, we use `VGG16` [SZ15] networks with ReLU activations and train them on CIFAR-10 [Kri09], a dataset containing RGB images of (pixel-, channel-) size $32 \times 32 \times 3$ with 10 classes (containing objects, such as automobiles, but also creatures, such as frogs or cats) from scratch. After training,[2] our network reaches an average accuracy of 82% on the test set. As stated above, we employ the layerwise coverage (LWC) notion defined by Abrecht et al. [AAG+20] to measure NC as it subsumes less granular metrics, such as the often-used definitions of DeepXplore [PCYJ19]. Subsumption implies that if DeepXplore does not reach full coverage, neither will LWC but not necessarily the other way around.

[2] We employ an SGD optimizer with a learning rate of 10^{-3} for 20 epochs and a learning rate decay of 0.1 at epochs 10 and 15.

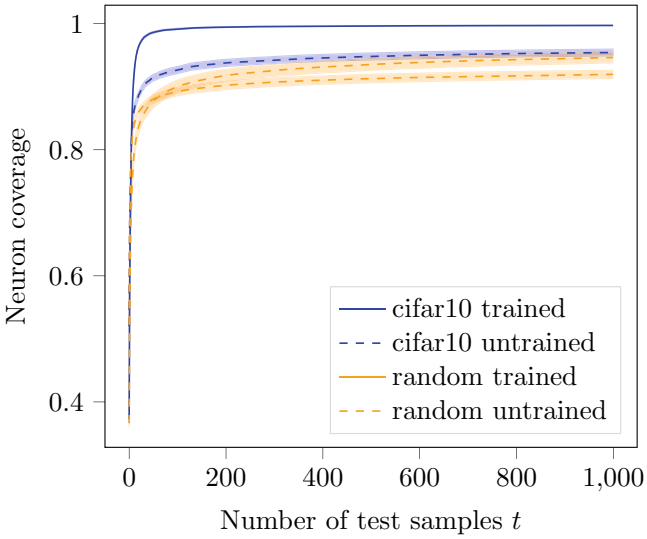

Fig. 1 Here, we show the neuron coverage in a `VGG16` network. The blue solid and dashed lines show mean coverage for trained and untrained models on CIFAR-10 test images, respectively. The orange solid and dashed lines show mean coverage for trained and untrained models on randomly generated data samples, respectively. The standard deviation is shown for each experiment as the shaded area

In Fig. 1, we plot the NC of the used `VGG16` DNNs with respect to 1000 test samples for all four test cases. For each case, we employed five different DNNs, independently initialized and/or trained, and averaged the results for further stability. For evaluation, we use both the real CIFAR-10 test set data and randomly generated data samples from a Gaussian distribution loosely fitting the overall data range. While the coverage behaves qualitatively similar in all cases, a clear separation between (assumed) limiting values and saturation speeds can be seen. This difference is most pronounced between the trained DNN that is tested with CIFAR-10 test data, representing the most structured case as both the data and the weights carry (semantic) meaning, and all other cases, which contain at least one random component.

A rough understanding of the behavior of NC can be acquired when considering the following highly simplified thought experiment. Assuming that the activation of a neuron was an entirely probabilistic event in which a given input sample activates the neuron with probability $0 < p < 1$, then the chance of said neuron to remain inactive for t samples would be q^t with $q := 1 - p$. While this suggests an exponential saturation rate for the coverage of a single neuron, the idea only carries over to the full set of all considered neurons if one assumes further that their activation probabilities p' were identical and their activations independent. Even omitting only the first condition (identicality) can lead to qualitatively different behavior, in which specific properties would depend on the distribution of p's among the neurons. A typical

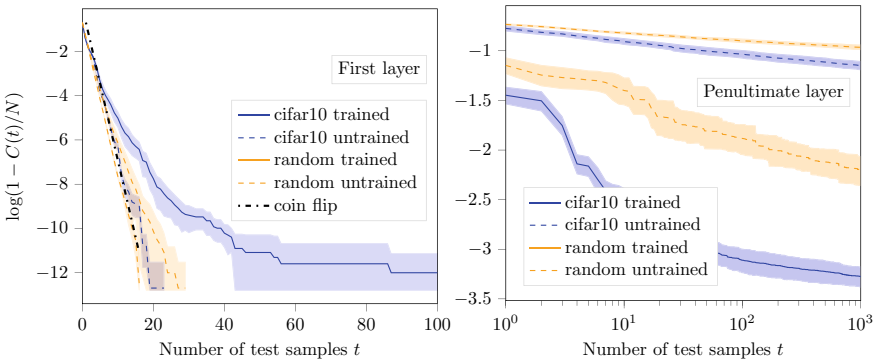

Fig. 2 Log plot of mean coverage in first (left) and penultimate (right) layer for trained and untrained networks. In the first layer, neuron coverage shows similar behavior for trained and untrained networks. However, in the penultimate layer, there is a significant difference in behavior between trained and untrained networks. The standard deviation is shown as shaded region. As we consider $\log(1 - C(t)/N)$, with the number of covered neurons $C(t)$, total neurons N and test samples t, the value approaches "negative infinity" when full coverage is achieved. This behavior can be seen in the figure on the left. The x-axis on the right is on log scale

expectation for a broad distribution would be an approximately algebraic saturation.[3] Unsurprisingly, we can recover both types of behavior within the experiments.

A connection to the above thought experiment is obtained when investigating NC layer-wise. While from the perspective of testing, one might argue that the coverage of any given layer is subsumed in the coverage of the full model, the differing behavior of the layers warrants investigating NC on this level of granularity. For the first and the penultimate layer of `VGG16`, the results are depicted in Fig. 2. In the first layer, one would expect that, at least for random input, each neuron is equally well suited to be activated, and thus saturation would be reached exponentially. As can be seen, this is almost ideally given for the untrained DNN if tested on random input, and in good approximation still when using the structured CIFAR-10 input or when testing the trained DNN with the random samples. In the latter case, the decision boundaries of each trained neuron are "random" with respect to the incoming data, which does not represent any meaningful semantic concepts or correlations. Note that we do not show the coverage in the figure but rather the number of not-yet-activated neurons in a logarithmic fashion; straight lines represent exponential saturation. Regarding the speed of saturation, i.e., the exponent or probability p, the naive assumption is that a neuron separates the input via a half-plane and each case, activation or inactivation, is equally likely. This corresponds to a coin-flip with probability $p = q = 1/2$ and is depicted in the figure for comparison. Only the trained DNN, when tested on actual test data, deviates from this behavior and saturates more slowly. This might be

[3] Consider, for comparison, how a superposition or mixture of Gaussians turns into a heavy-tailed distribution, e.g., Student's t, if the standard deviations of each Gaussian differ strongly enough, cf. [Bar12].

explained by the fact that the network can use feature-based correlation in the data, and a new test sample is therefore of a more limited novelty than a fully random, and thus unexpected one.

Turning to the penultimate layer, shown on the right-hand side in Fig. 2, we choose a double-logarithmic visualization such that algebraic saturation is now given by straight lines. To some extent, all four test cases seem to satisfy this type of saturation. However, the exponent, i.e., the speed of saturation, differs significantly and is slowest for both cases of the untrained DNN. A possible explanation might simply be the propagation of information that is not optimized in a random DNN. By design, a (classification) DNN maps a large input (here 3072-dimensional) onto a low-dimensional output (10 classes) and, thus, has to discard large amounts of information along the layers. This heuristic assumption is supported by the behavior of the trained DNN, which achieves high coverage significantly faster, even more so if the input contains features it was optimized to recognize (i.e., actual test data).

As shown, NC does depend on the training and the use of an appropriate test set. Even then, the final coverage might be reached only slowly. However, returning to the original Fig. 1, relatively high coverage is reached early on with only a few samples. It is, therefore, more a question of resolving a "hand-full" of not-covered neurons that might not be receptive to the data used. In the next section, we will look more closely into whether or not all neurons can even be covered if the used data are restricted to the operational domain of the network. This property, catching large amounts of the coverage with only limited effort, might not be desirable from the perspective of testing, especially if the novelty of samples is supposed to be judged. For such use cases, more granular tests might be more appropriate. A straightforward extension might be to include not only "positive" neuron coverage (NC+) as done before but also "negative" coverage (NC-) where a neuron was not activated at least once. For a neuron, it might be the case that it is almost always active, and therefore the rare cases where it does not fire might be of interest. For this reason, both types of coverage are to some degree independent, i.e., while it can be possible for a DNN to reach full NC+ coverage, this does not necessarily ensure full (NC-) coverage, and vice versa. Especially for networks with ReLU activations, (NC-) coverage is of interest as the non-linearity only takes effect when a neuron is not active. For a discussion, see, for instance, [EP20].

As discussed in the introduction, a further logical extension could be a switch from measuring single neuron activations or inactivations to a notion of pairwise coverage (PWC). Here, we follow a simplified version of the sign-sign coverage introduced in [SHK+19]. Due to the typically feed-forward type of network structure, we limit ourselves to adjacent layers with a pair of neurons as the unit of interest. A pair of neurons, $n_{\ell,i}, n_{\ell+1,j}$ from layers ℓ and $\ell+1$, counts as covered with respect to PWC++, if for a given input both neurons are active, i.e., assuming values greater than 0. Similarly, we define PWC−− for cases where the pre-activations are both non-positive, and the neuron outputs thus zero. Likewise, the two alternating activation patterns, PWC+– and PWC–+, can be defined. It is easy to see that if a DNN is fully PWC++ (or PWC−−) covered, it is also fully NC+ (NC–, respectively) covered. Therefore, the PWC metrics subsume the single neuron NC metrics. However, this

Table 1 NC and PWC: This table shows the big difference between the number of countable units of neurons in NC and pairwise evaluation in PWC. Neurons are defined as per LWC [AAG+20]. For simplicity, we do not consider the effect of skip connections

Network	NC [10^5]	PWC [10^5]
`LeNet-5` [LBBH98]	0.081	81
`VGG-16` [SZ15]	3.159	82,030
`VGG-19` [SZ15]	3.425	85,428
`ResNet-18` [HZRS16]	0.548	1,673
`ResNet-34` [HZRS16]	0.804	2,265
`ResNet-50` [HZRS16]	2.309	12,117
`ResNet-101` [HZRS16]	3.354	13,721

increased granularity comes at a price. While the number of tests or conditions required to be fulfilled to reach full NC coverage scales linearly with the number of neurons or units, pairwise metrics follow a quadratic growth. Due to the restriction to neighboring layers, the growth is effectively reduced by the number of layers, and the number of conditions can be approximated by $(\#\text{neurons})^2/\#\text{layers}$. Exact numbers for a selected few architectures are reported in Table 1 including the `VGG16` example used so far. Note that for simplicity, skip connection combinations, although forming direct links between remote layers, are not considered. Including them into the pairwise metric would lead to larger numbers. Even without, the sheer magnitude shows that it might be unlikely to test for full coverage. Additionally, it is unclear whether full coverage can be reached in principle if the neuron output is correlated across layers. As this might pose a problem for tests that require an end criterion signifying when the test is finished, we include those more elaborate metrics in the next section where we investigate whether full coverage can be reached.

4 Formal Analysis of Achievable Coverage

In this section, we analyze coverage metrics based on a very simple task and a simple domain as described in Sect. 4.1. The key design objectives for the creation of the task are that (i) arbitrary many data samples can be generated, (ii) a ground truth oracle exists in closed form, and (iii) the trained network and the input domain are sufficiently small such that specific activation patterns can be computed analytically. We describe the experimental setup in detail in Sect. 4.2, before providing the experimental results in Sect. 4.3, followed by a discussion in Sect. 4.4.

4.1 Description of the Task

Our simple task is counting the number of input "pixels" with positive values. We restrict ourselves to a system with four input pixels, whose values are constrained to the interval $\mathcal{I} = [-1, 1]$. We call this the operational design domain (ODD) of our network, following [KF19]. This is also our training distribution. We can obviously also perform inference on samples outside this ODD, but these will be outside the training distribution since we only train on ODD samples. We frame this task as a classification problem with five output classes: the integers from 0 to 4. Thus, our input domain is $\mathcal{I}^4$, our output domain is $\mathcal{O} = \{0, 1, 2, 3, 4\}$, and we can define a ground truth oracle as cardinality $|\{i | i \in \mathcal{I}, i > 0\}|$, i.e., we count the pixels with a value above 0. In addition, we explicitly define all inputs in $\mathbb{R}^4 \setminus \mathcal{I}^4$ as out-of-distribution samples, with $\mathcal{I}^4 \subset \mathbb{R}^4$.

The rationale of our approach is as follows: What can be achieved with perfect knowledge of the system in the sense of (*i*) symbolically computing inputs to achieve activation patterns, (*ii*) having a clear definition of in-ODD and out-of-ODD, and (*iii*) having a perfect labeling function for novel examples?

For this task, we use a multilayer perceptron (MLP) with three hidden fully connected layers FC(n) with n output nodes. The architecture of the MLP is shown in Fig. 3. We train our network using `PyTorch`. We generate one million inputs uniformly sampled from $\mathcal{I}^4$. We split these samples into training and test data sets

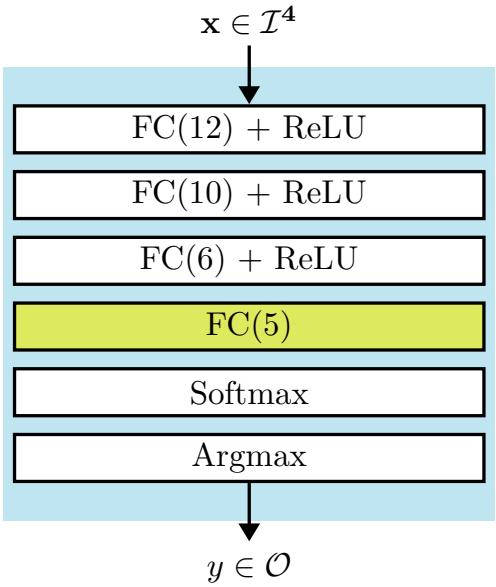

Fig. 3 Architecture of the MLP

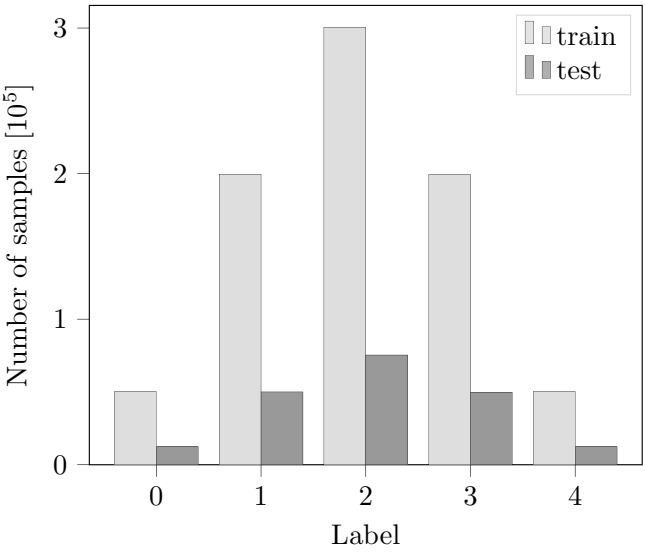

Fig. 4 Distribution of labels in training and test set, relative distribution is identical for both

Table 2 Accuracy [%] overall and per class for MLP_{adv} and $\text{MLP}_{\text{plain}}$. Best numbers are in boldface. Performance of $\text{MLP}_{\text{plain}}$ is slightly better, but differences are insignificant

Network	Overall	0	1	2	3	4
MLP_{adv}	99.54	**99.94**	99.64	99.16	99.83	99.89
$\text{MLP}_{\text{plain}}$	**99.89**	99.87	**99.83**	**99.88**	**99.96**	**99.97**

using an 80/20 split, i.e., we end up with 800,000 training samples and 200,000 test samples. The distribution of the respective ground truth labels is shown in Fig. 4. The distribution of the samples is as expected as there are 16 unique combinations of positive and negative pixels in $\mathcal{I}$ of which 6 yield two positive pixels, 4 yield one or three positive pixel(s), respectively, and 1 combination yields 0 or 4 positive pixels, respectively.

In the following, we study the impact of coverage models and the ODD on specific networks to show specific effects rather than a statistical evaluation. Nevertheless, we show that this is not a singular effect by training the MLP, whose architecture is depicted in Fig. 3, with two training setups, leading to two different MLPs, MLP_{adv} and $\text{MLP}_{\text{plain}}$. We train both MLPs for 100 epochs using ADAM as an optimizer with default `PyTorch` settings, a batch size of 100, standard cross-entropy loss, and a single initialization. For increasing robustness, MLP_{adv} leverages adversarial training, here with the fast gradient sign method (FGSM) [GSS15], using $\epsilon = 10^{-4}$, which corresponds to the attack strength. For $\text{MLP}_{\text{plain}}$, we use the same training setup but do not apply adversarial training. Table 2 summarizes the final performances on the test set for the overall accuracy and the per-class accuracy for both networks.

We can see that $\text{MLP}_{\text{plain}}$ has a slightly higher accuracy overall, as well as for four out of the five output classes, than MLP_{adv}. Note that the lower accuracy of MLP_{adv} is in line with standard robust training and shows that when trying to train a robust network, more capacity may be necessary [GSS15]. We focus in the following on the adversarially trained model for the sake of brevity since both setups resulted in similar results.

4.2 Experimental Setup

In our experiments, we want to evaluate whether full coverage according to neuron coverage (NC) and pairwise coverage (PWC) can be achieved. The networks for the experiments are the trained MLPs described in the previous section. The goal of the experiment is to calculate a minimal number of inputs that maximizes coverage on the MLP and, if required, a number of non-coverable items.

Since the MLP is simple enough, we can solve the coverage generation symbolically. For this, we use a satisfiability modulo theory (SMT) solver, namely, `Z3` [dMB08], and encode the problem of finding inputs maximizing coverage as SMT queries based on the theory of unquantified linear real arithmetic, i.e., Boolean combinations of inequations between linear polynomials over real variables. Note that there are also further approaches based on SMT targeting larger neural networks like `Reluplex` [KBD+17], which are, however, not needed for our example. We iteratively compute solutions for individual neurons (or combinations of neurons) as shown in Algorithm 2: We determine all coverage items for a neural network as a set $\mathcal{B}$. While there are still coverage items, we select one of these and try to cover it as described in the following. We formulate the coverage of item $i \in \mathcal{B}$ as an SMT query and call the SMT solver on this query using `runSMT()`. If it succeeds as indicated by the return value *success* being true, we run the network `NN` on the resulting input $\mathbf{x}$ by executing `callMLP()`. This returns all resulting coverage items $\mathcal{J}$, which are added to the set of already covered coverage items $\mathcal{C}$. Minimally, we find the coverage item i from the SMT query, but there may be more. If the SMT solver cannot cover item i, we add it to the set of uncoverable items $\mathcal{U}$. Note that this implicitly also depends on a specific coverage metric, which we omit in the algorithm for simplicity of the presentation.

The function `runSMT()` generates individual samples based on a program for `Z3` [dMB08], or `Z3Py`[4], respectively, as a list of constraints. These constraints encode the network, the input domain, and the coverage item. When the SMT determines the constraint system as satisfiable, it returns a model. From this model, we can extract the corresponding inputs to the network that lead to a satisfiable result.

For each coverage item, we run two experiments with and without a constrained input domain. First, we use constraints as shown above that allow the `Z3` solver to only consider in-ODD samples from $\mathcal{I}$[4]. Second, we also allow `Z3` to generate

[4] https://pypi.org/project/z3-solver.

Algorithm 2 High-Level Algorithm

```
1: procedure ComputeSMTCoverage(NN)                ▷ NN is the network to be covered
2:     B ← getCoverageItems(NN)
3:     C ← ∅                                       ▷ items already covered
4:     U ← ∅                                       ▷ uncoverable items
5:     while B \ {C ∪ U} ≠ ∅ do
6:         Select i ∈ B \ {C ∪ U}
7:         success, x ← runSMT(NN, i)              ▷ x is an input for NN
8:         if success then
9:             J ← callMLP(x)                      ▷ compute all possible coverage items for x
10:            C = C ∪ J
11:        else
12:            U = U ∪ {i}
13:        end if
14:    end while
15: end procedure
```

arbitrary samples from $\mathbb{R}^4$, i.e., we allow `Z3` to also generate out-of-ODD samples. Naturally, the achievable coverage using only in-ODD samples can be at most as high as the coverage for the entire $\mathbb{R}^4$.

As a baseline, we compare the coverage achieved by the samples from `Z3` with two sampling-based approaches: (*i*) We use random sampling from the input domain (further described below) since it is a commonly applied strategy for data acquisition. (*ii*) Due to the compactness of our input space, we can use a grid-based approach of the input domain. We include this comparison to check how easy or difficult it is to achieve maximum coverage using sampling-based approaches. In particular, we use a grid-based sampling with a fixed number of samples per dimension, and we sample at random from a distribution. As for the structured generation, we perform two experiments for each approach: One permitting only in-ODD samples from $\mathcal{I}^4$, and one also permitting out-of-ODD samples. Of course, since we know the exact maximum coverage for each of the considered coverage metrics from `Z3`, we are only interested in how far below the optimum the sampling-based approaches remain.

For the grid-based approach, we sample from $\mathcal{I}^4$ with a resolution of 100 equidistant samples per dimension for the in-ODD samples. In addition, we sample from $\mathcal{R}^4 \subset \mathbb{R}^4, \mathcal{R} = [-2, 2]$ with a resolution of 100 equidistant samples per dimension for a mixture of in-ODD and out-of-ODD samples. For the random sampling approach, we sample 10^8 data points uniformly at random from $\mathcal{I}^4$ for in-ODD samples. In addition, we sample 10^8 data points from a Gaussian distribution with zero mean and unit variance, which results in 78% out-of-ODD samples. For all data sets, we measure coverage based on `PyTorch` [PGM+19] as described above in Algorithm 2 for the SMT-based approach.

4.3 Experimental Results

Table 3 shows the number of generated samples, whereas Table 4 summarizes the coverage results from our experiments, both for MLP_{adv}. Please note that these results are for a single network. However, we are not interested in the specific numbers here but rather compare different coverage models and data generation approaches qualitatively on a specific network. The structure of Table 4 is as follows: Each row contains the results for one data generation approach. Thus, the first row contains the result when optimizing for positive neuron coverage (NC (+)), i.e., getting a positive activation for each individual neuron. The second row contains the results for neuron coverage in both directions (NC (both)), i.e., each individual neuron has at least once a positive and a negative activation. The third row contains the results when optimizing PWC. The final two rows contain the results for the two baseline sampling approaches.

The columns of the table show the resulting coverage for the different coverage metrics. For each column, we provide separate columns *All* and *ODD*: The *ODD* columns contain the results when only in-ODD samples are used, whereas the *All* columns contain the result when also allowing for out-of-ODD samples. For NC (+) and NC (both), the cells contain the coverage results for each of the three layers, denoted by L1 to L3. Please note that L1 is the layer directly after the input and L3 constructs the representation, which is used in the final classification layer L4. For PWC, we further split the *All* and *ODD* columns based on the layer combinations that are considered. The L1–L2 columns contain the results for the coverage of the interaction between the neurons on layers L1 and L2 while the L2–L3 columns contain the results for the interaction between L2 and L3, respectively. In these columns, the entries in the single cells refer to the coverage of the four possible combinations of activations for the considered pairs of neurons. As an example, in column L1-L2, "$+-$" refers to all cases where a neuron in L1 has a positive activation, and a neuron in L2 has a negative activation. In each row, we marked the cells with gray background where the reported coverage corresponds to the optimization target in `Z3`. These entries signify the maximally possible coverage values.

In the following, we highlight some interesting aspects of the results. Note that, in general, the coverage in the *All* column must be larger or equal to the coverage in the *ODD* column for each of the coverage metrics as $\mathbb{R} \supset \mathcal{I}$.

Intuitively, we would expect that the more elaborate a coverage metric is (i.e., the more elements it requires to be covered), the more test samples are required to achieve coverage. In our results, this expectation is satisfied, i.e., we need more tests

Table 3 Number of tests per approach

	NC (+)	NC (both)	PWC	Grid	Sample
All	6	10	39	10^8	10^8
ODD	5	10	47	10^8	10^8

Table 4 Results from the coverage analysis, Lℓ means layer ℓ. The rows show different data generation approaches (solver-based and sampling-based). The columns show coverage based on different coverage metrics. Grey shaded cells show the results of metric that was optimized in SMT

Data Generation Approach	Results for different coverage metrics							
	NC (+)		NC (–)		PWC			
	All	ODD	All	ODD	All		ODD	
					L1-L2	L2-L3	L1-L2	L2-L3
NC (+)	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 0.80 L3: 0.83	L1: 1.00 L2: 0.90 L3: 1.00	L1: 0.50 L2: 0.60 L3: 0.50	++: 0.85 +−: 0.81 −+: 0.62 −−: 0.69	++: 0.82 +−: 0.72 −+: 0.83 −−: 0.73	++: 0.67 +−: 0.56 −+: 0.34 −−: 0.26	++: 0.57 +−: 0.37 −+: 0.47 −−: 0.27
NC (both)	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 0.80 L3: 0.83	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 1.00 L3: 1.00	++: 0.89 +−: 0.84 −+: 0.79 −−: 0.77	++: 0.92 +−: 0.80 −+: 0.78 −−: 0.77	++: 0.78 +−: 0.81 −+: 0.65 −−: 0.68	++: 0.67 +−: 0.72 −+: 0.82 −−: 0.53
PWC	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 0.80 L3: 0.83	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 1.00 L3: 1.00	++: 0.98 +−: 1.00 −+: 0.97 −−: 1.00	++: 0.95 +−: 0.95−+: 0.98 −−: 0.97	++: 0.80 +−: 0.94 −+: 0.78 −−: 0.88	++: 0.67 +−: 0.77 −+: 0.82 −−: 0.77
Grid	L1: 1.00 L2: 1.00 L3: 0.83	L1: 1.00 L2: 0.80 L3: 0.83	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 1.00 L3: 1.00	++: 0.97 +−: 0.99 −+: 0.96 −−: 0.95	++: 0.82 +−: 0.92 −+: 0.82 −−: 0.88	++: 0.79 +−: 0.93 −+: 0.76 −−: 0.88	++: 0.67 +−: 0.73 −+: 0.82 −−: 0.75
Sample	L1: 1.00 L2: 1.00 L3: 0.83	L1: 1.00 L2: 0.80 L3: 0.83	L1: 1.00 L2: 1.00 L3: 1.00	L1: 1.00 L2: 1.00 L3: 1.00	++: 0.98 +−: 1.00 −+: 0.96 −−: 0.99	++: 0.82 +−: 0.92 −+: 0.83 −−: 0.93	++: 0.80 +−: 0.94 −+: 0.78 −−: 0.88	++: 0.67 +−: 0.75 −+: 0.82 −−: 0.77

for PWC than for NC, and we need more tests for NC (both) than for NC (+) as shown in Table 3. In addition, we would expect that coverage is proportional to the number of tests for the first three rows, i.e., more tests imply higher coverage, given that we only add new tests if they actually increase coverage. Interestingly, this is not satisfied for PWC, where we generate more tests for in-ODD samples than for *All*, even though the resulting coverage for *ODD* is lower than for *All*. It seems that in our particular MLP, it is easier to stimulate multiple neurons at once when using out-of-ODD samples.

Let us now consider NC. When using arbitrary test samples, it is possible to achieve 100% coverage for both NC (+) and NC (-). However, when restricting the generation to only in-ODD samples, i.e., a subset of the complete input space of the MLP, full coverage can no longer be reached for NC (+). In particular, we observe that some neurons on layers L2 and L3 can no longer be stimulated to have a positive

activation. Considering the use of SMT leading to this observation, we thus not only have empirical evidence but also a tool-based mathematical proof that 100% coverage is not possible in this case.[5]

If you compare the coverage in the NC (+) column with the corresponding coverage for the two sampling-based approaches, you will notice that they achieve the same non-100% coverage for in-ODD samples and a smaller non-100% coverage for *All* samples. Thus, the sampling-based approaches failed to achieve 100% coverage for NC (+) in both cases. While we know in this example that no better result was theoretically possible for in-ODD samples, more coverage could have been possible for *All*. However, for a more complex task, these two cases cannot be distinguished, i.e., if we have coverage less than 100%, we cannot decide whether it is impossible to achieve more or whether our test set is yet incomplete. In addition, the result shows that even in this simple domain, a relatively high number of samples do not achieve maximal coverage. Thus, it is highly unlikely to achieve the theoretically possible coverage with respect to more involved metrics such as PWC on complex domains in computer vision using random or grid-based sampling.

As stated above, we can achieve higher coverage when also allowing for out-of-ODD samples. In our example, the network was not trained on such out-of-ODD samples, so the question remains whether it is actually meaningful to include such samples in a test set just for increasing coverage. In our example, we can give some intuition on this. For this, we look in detail at samples generated by our SMT-based approach with and without input constraints. The results are shown in Fig. 5, where we plot samples SMT generated for the NC (both) criterion. Each run creates 10 samples, as is shown in Table 4. Each of these samples is labeled according to our labeling function, and the corresponding cross-entropy loss J^{CE} is computed. Since the loss varies drastically, we show a logarithmic plot and since the loss can be zero for perfect predictions, we specifically plot

$$\log(J^{\text{CE}} + 1). \tag{1}$$

Remember from Table 3 that there are 10 samples, each for the full input space and the ODD space. As we can see, for both input domains, for the first three samples, the network generates perfect predictions and therefore $\log(J^{\text{CE}} + 1) := 0$. For the ODD samples, we can see that the loss is quite low and at most in the order of the maximum training loss (as indicated by the dotted line). We also annotate for each sample of the full input space (All) whether the resulting sample is in the ODD, labeled as All (in) and indicated by a dot, or whether the sample is out-of-ODD, labeled as All (out) and indicated by a diamond shape. As we can see, when optimizing coverage without constraints (All), most of the samples (7/10) produced by `Z3` are out-of-ODD, even though also in-ODD samples would exist for the most part. In the figure, we can see that these out-of-ODD samples are not suitable for a realistic evaluation: Each out-of-ODD sample has a very high loss, at least an order of magnitude higher than

[5] Please note that the large-scale experiments with random and grid-based sampling provide additional empirical evidence.

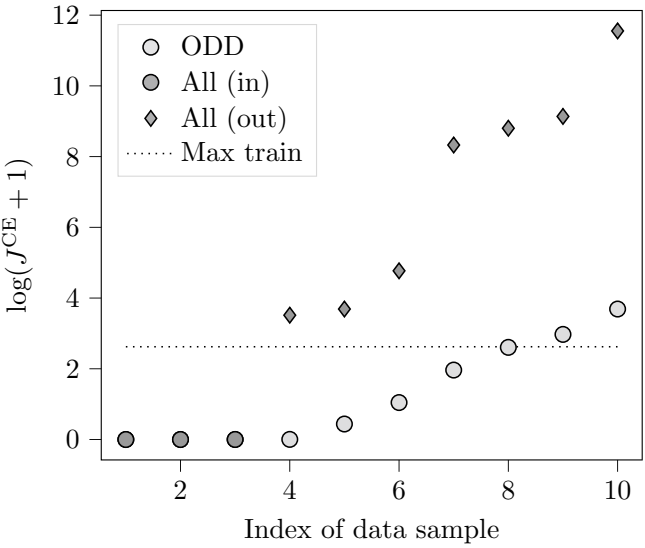

Fig. 5 Log-modified cross-entropy losses (see (1)) on samples generated by SMT for NC (both). ODD refers to in-ODD samples, All (in/out) refer to in-ODD/out-of-ODD samples sampled from a mixed in-ODD/out-of-ODD input space, and Max train is the maximum training loss

the maximum training loss, marked by a dotted line. It can also be observed that in-ODD samples have a significantly lower loss. Nevertheless, even here, we see that the SMT-based approach generates two very difficult examples.

Intuitively, we would expect that achieving a high coverage is more difficult on deeper layers because achieving a specific activation also requires establishing necessary activations on the previous layers. This means these previous layers work like constraints for the computation of an input that creates an activation on a deeper layer. Thus, the deeper the layer, the more constraints need to be satisfied. While our results confirm this intuition in most cases, these are not as strong as you can see, e.g., when comparing the coverage for PWC. Here, the coverage for L2–L3 is only slightly lower in three out of four cases than for L1–L2, and in the fourth case, it is even higher for the deeper layers. This is most probably the case because the network is rather shallow.

Note that we additionally performed these experiments on $\mathrm{MLP}_{\mathrm{plain}}$. While the results are largely the same in direction (reductions), the magnitude of differences is slightly larger. We hypothesize that this is due to the lower robustness of representations and, thus, preferred to show the results for the adversarially trained MLP.

4.4 Discussion

Our results for the formal analysis of achievable coverage clearly show that even for simple networks, 100% coverage may not be achievable. This has two reasons: (*i*) Networks are trained in a way that they contain some activations that may only ever be used in one direction. We can see this in the *All* columns of Table 4 that have coverage levels $< 100\%$. In traditional software, we would refer to this as "dead code" that results in non-perfect coverage. One could argue that as with "dead code" in traditional software, there may be a way to remove this by some form of static analysis. (*ii*) Even if we remove this "static dead code", we can still see a difference between *All* and *ODD*. This difference is harder to identify in practice as—in contrast to our simple example—this distinction between *All* and *ODD* is not always obvious for computer vision tasks. Hence, this is a harder form of "dead code" depending on the particular input/operational distribution. Both of these sources result in incomplete coverage.

Let us summarize major differences between state-of-the-art DNNs used, e.g., in computer vision for automated driving functions, and our presented MLP example:

- In a complex input domain, we do not have a clear definition of ODD and All, and we may not be able to decide whether a specific input is reasonable or not. For example, it is not clear whether a new image lies in the challenging part of the ODD or is considered to lie outside the ODD.
- We usually want to focus on in-ODD examples.
- Formal approaches such as the one used in these examples do not scale to typical state-of-the-art DNNs. Note that we cannot directly use more efficient, approximate works, e.g., on adversarial examples, since they perform an over-approximation while we are interested in under-approximations for coverage.

As a result, the identification of both sources of "dead code" is typically infeasible. This means that we do not know the real upper threshold for achievable coverage and must over-approximate it with the complete number of coverage items, including both sources of "dead code".

5 Conclusion

In summary, while neuron coverage remains promising in certain aspects for V&V, it comes with many caveats. The granularity of the metric in use determines how effective the metric is at uncovering faults. The granularity of application of the metric, layerwise instead of full model, also reveals further interesting behavior of the model under test. Similarly, when studying the novelty of input samples, we saw that random inputs show more novelty and lead to more coverage due to their unexpected nature. Structured input with a trained network has lower novelty in comparison. While test generation using NC focuses on generating realistic novel

samples, it is unclear if the increase in coverage from these generated samples is due to some random noise injected into the generated image or meaningful semantic change.

Coverage metrics are also hard to interpret if the metric converges to a value below 100%. This means if we determine that a certain neuron is not coverable, we cannot determine (*i*) whether it is coverable with *some* input, and moreover (*ii*) with some input inside our ODD. Note that, depending on the ODD, using an adversarial example in some ϵ-environment may not be reasonable for the ODD. Consequently, the coverage metrics are not actionable in the sense that we can derive information on which particular inputs we lack in our test set.

As we have shown, filling the test set with out-of-ODD samples for increasing coverage has a questionable effect. Since they are out-of-ODD, the network under test will usually not be trained on such samples. As a result, the network needs to extrapolate for classifying them and yields a high loss in our samples in Fig. 5. However, in real tasks, we do not necessarily know whether the newly generated data is out of the ODD, and the high loss may indicate that such samples should further be considered, e.g., added to the training set.

Acknowledgements The research leading to these results is funded by the German Federal Ministry for Economic Affairs and Energy within the project "Methoden und Maßnahmen zur Absicherung von KI-basierten Wahrnehmungsfunktionen für das automatisierte Fahren (KI Absicherung)". The authors would like to thank the consortium for the successful cooperation.

References

[AAG+20] S. Abrecht, M. Akila, S.S. Gannamaneni, K. Groh, C. Heinzemann, S. Houben, M. Woehrle, Revisiting neuron coverage and its application to test generation, in *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP) Workshops* (Virtual Conference, 2020), pp. 289–301

[AGG+21] Stephanie Abrecht, Lydia Gauerhof, Christoph Gladisch, Konrad Groh, Christian Heinzemann, Matthias Woehrle, Testing deep learning-based visual perception for automated driving. ACM Trans. Cyber-Phys. Syst. **5**(4), 1–28 (2021)

[Bar12] D. Barber, *Bayesian Reasoning and Machine Learning* (Cambridge University Press, 2012)

[BEW+18] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, J. Törnqvist, Safely entering the deep: a review of verification and validation for machine learning and a challenge elicitation in the automotive industry, pp. 1–29 (2018). arXiv:1812.05389

[dMB08] L. Mendonça de Moura, N. Bjørner. Z3: an efficient SMT solver, in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Budapest, Hungary, 2008), pp. 337–340

[DZW+19] Y. Dong, P. Zhang, J. Wang, S. Liu, J. Sun, J. Hao, X. Wang, L. Wang, J.S. Dong, D. Ting, There is limited correlation between coverage and robustness for deep neural networks, pp. 1–12 (2019). arXiv: 1911.05904

[EP20] T. Ergen, M. Pilanci, Convex geometry of two-layer ReLU networks: implicit autoencoding and interpretable models, in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (Virtual Conference, 2020), pp. 4024–4033

[GSS15] I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in *Proceedings of the International Conference on Learning Representations (ICLR)* (San Diego, CA, USA, 2015), pp. 1–11

[HCWG+20] F. Harel-Canada, L. Wang, M.A. Gulzar, Q. Gu, M. Kim, Is neuron coverage a meaningful measure for testing deep neural networks? in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE))* (Virtual Conference, 2020), pp. 851–862

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, NV, USA, 2016), pp. 770–778

[KBD+17] G. Katz, C. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: an efficient SMT solver for verifying deep neural networks, in *Proceedings of the International Conference on Computer Aided Verification (CAV)* (Heidelberg, Germany, 2017), pp. 97–117

[KF19] P. Koopman, F. Fratrik, How many operational design domains, objects, and events? in *Proceedings of the Workshop on Artificial Intelligence Safety (SafeAI)* (Honolulu, HI, USA, 2019), pp. 1–8

[Kri09] A. Krizhevsky, *Object Classification Experiments* (Technical report, Canadian Institute for Advanced Research, 2009)

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. IEEE **86**(11), 2278–2324 (1998)

[MJXZ+18] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, Y. Wang, DeepGauge: multi-granularity testing criteria for deep learning systems, in *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE)* (Montpellier, France, 2018), pp. 120–131

[PCYJ19] Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana, DeepXplore: automated whitebox testing of deep learning systems. Commun. ACM **62**(11), 137–145 (2019)

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, et al., PyTorch: an imperative style, high-performance deep learning library, in *Proceedings of the Conference on Neural Information Processing Systems (NIPS/NeurIPS)* (Vancouver, BC, Canada, 2019), pp. 8024–8035

[RJS+20] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, Paolo Tonella, Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. **25**(6), 5193–5254 (2020)

[SHK+19] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, Rob Ashmore, Structural test coverage criteria for deep neural networks. ACM Trans. Embed. Comput. Syst. (TECS) **18**(5s), 1–23 (2019)

[SZ15] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *Proceedings of the International Conference on Learning Representations (ICLR)* (San Diego, CA, USA, 2015), pp. 1–14

[TPJR18] Y. Tian, K. Pei, S. Jana, B. Ray, DeepTest: automated testing of deep-neural-network-driven autonomous cars, in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)* (2018), pp. 303–314

[ZHML20] J.M. Zhang, M. Harman, L. Ma, Y. Liu, Machine learning testing: survey, landscapes and horizons. IEEE Trans. Softw. Eng. **48**(01), 1–36 (2022). https://doi.org/10.1109/TSE.2019.2962027